Nginx完整教案

Nginx的安装

1、下载Nginx

wget http://nginx.org/download/nginx-1.14.2.tar.gz （stable）

2、安装Nginx依赖

nginx是C语言开发，建议在linux上运行，本教程使用**Centos7.4**为安装环境。

yum install -y gcc-gcc++ pcre pcre-devel zlib zlib-devel openssl openssl-devel

3、解压安装Nginx

解压

tar xf nginx-1.14.2.tar.gz

进行configure配置

./configure --prefix=/usr/local/nginx （可以配置很多nginx其他模块，后面配置https的时候讲解）

编译和安装

make & make install -j 4

如果以上操作没有报错则安装成功

启动nginx

./nginx -c 指定配置文件位置

eg: ./nginx -c /usr/local/conf/nginx.conf

./nginx 默认使用NGINX_HOME/config/nginx.conf配置文件

停止niginx

./nginx -s stop 停止

./nginx -s quit 退出

./nginx -s reload 重新加载nginx.conf （很常用）

发送信号量 （找不到nginx安装位置，但是想要停止nginx服务器情况下使用）

kill -TERM master进程号

kill -QUIT master进程号

4、nginx 常用命令

./nginx -v 查看nginx版本

./nginx -V 查看nginx的编译版本及配置的参数

./nginx -t 主要验证nginx.conf配置文件是否有问题

./nginx -c 根据配置文件的位置启动nginx

./nginx -s 发送对应信号处理master进程

-s signal   : send signal to a master process: stop, quit, reopen, reload

基本配置文件

1、基本结构

```
// nginx全局块
...

// events块
events {
    ...
}

// http 块
http {
    // http全局块
    ...

    // server块
    server {
        ...
    }

    // http全局块
    ...
}
// upstream 块
upstream {

}
```

2、配置文件详解

2.1 nginx.conf配置文件

```
# 配置nginx的用户组 默认为nobody
#user  nobody;

#指定工作进程的个数，默认是1个，具体可以根据服务器cpu数量进行设置，如果不知道cpu的数量，可以设置为auto
```

```
worker_processes  1;

# 配置nginx的错误日志 格式为 log路径 log级别
# error_log 的日志级别为： debug info notice warn error crit alert emerg 紧急由低到高
# error_log的默认日志级别为error，那么就只有紧急程度大于等于error的才会记录在日志
# error_log 的作用域为 main http mail stream server location

#error_log  logs/error.log;
#error_log  logs/error.log  notice;
#error_log  logs/error.log  info;

# 指定nginx进程运行文件存放地址
#pid        logs/nginx.pid;

#工作模式及连接数上限
events {
    #参考事件模型，use [ select | poll | kqueue | epoll | rtsig | /dev/poll | eventport ]
    # poll是多路复用IO中的一种方式,但是仅用于linux2.6以上内核,可以大大提高nginx的性能
    # use epoll

    # 设置网络的连接序列化 防止惊群现象发生 默认为 on
    # accept_mutex on;

    # 设置一个进程是否同时接受多个网络连接 默认为 off
    # multi_accept off

    # 最大连接数 默认为 512
    worker_connections  1024;
    ########################################################################
    # 并发总数是 worker_processes 和 worker_connections 的乘积
    # 即 max_clients = worker_processes * worker_connections
    # 在设置了反向代理的情况下，max_clients = worker_processes * worker_connections / 4
    # 为什么上面反向代理要除以4，应该说是一个经验值
    # 根据以上条件，正常情况下的Nginx Server可以应付的最大连接数为：4 * 8000 = 32000
    # worker_connections 值的设置跟物理内存大小有关
    # 因为并发受IO约束，max_clients的值须小于系统可以打开的最大文件数
    # 而系统可以打开的最大文件数和内存大小成正比，一般1GB内存的机器上可以打开的文件数大约是10万左右
    # 我们来看看360M内存的VPS可以打开的文件句柄数是多少：
    # $ cat /proc/sys/fs/file-max
    # 输出 34336
    # 32000 < 34336，即并发连接总数小于系统可以打开的文件句柄总数，这样就在操作系统可以承受的范围之内
    # 所以，worker_connections 的值需根据 worker_processes 进程数目和系统可以打开的最大文件总数进行适
当地进行设置
    # 使得并发总数小于操作系统可以打开的最大文件数目
    # 其实质也就是根据主机的物理CPU和内存进行配置
    # 当然，理论上的并发总数可能会和实际有所偏差，因为主机还有其他的工作进程需要消耗系统资源。
}


http {
    # 文件扩展名和文件类型映射表
    include       mime.types;
```

```
    # 默认文件类型
    default_type  application/octet-stream;

    # 日志格式 后面会介绍
    #log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
    #                  '$status $body_bytes_sent "$http_referer" '
    #                  '"$http_user_agent" "$http_x_forwarded_for"';

    // 允许通过日志配置
    #access_log  logs/access.log  main;

    # sendfile 指定使用 sendfile 系统调用来传输文件。优点在于在两个文件描述符之间传递数据（完全在内核中操作），从而避免了数据在内核缓冲区和用户缓冲区之间的拷贝，效率高，称之为零拷贝
    # sendfile 作用域 location server http
    sendfile        on;

    #减少网络报文段的数量
    #tcp_nopush     on;

    # 链接超时时间 默认 75s 作用域 http server location
    #keepalive_timeout  0;
    keepalive_timeout  65;

    # 开始gzip压缩，降低带宽使用和加快传输速度，但增加了CPU的使用
    #gzip  on;

    server {
        # 端口号
        listen       8080;
        # 域名或ip
        server_name  localhost;

        #charset koi8-r;

        #access_log  logs/host.access.log  main;

        # 对请求的路由进行过滤 正则匹配
        location / {
            root   html;
            index  index.html index.htm;
        }

        ...
    }
    include servers/*;
}
```

3、Nginx 日志介绍

nginx的日志大致分为 `access_log` 和 `error_log`。error_log 记录的是nginx的错误日志。（以下对日志的理解不是很全面，还只是基础的）

3.1 error_log

- 记录nginx错误日志
- 作用域为 `main http mail stream server location`
- 日志级别 `debug info notice warn error crit alert emerg`
- 日志级别默认为 error 当级别高于或等于指定级别时才会记录

3.2 access_log

- 记录请求通过的日志
- 作用域为 `http server location limit_except`
- 日志格式默认为 `combined`
- 日志格式是可以自定义的

```
# 定义一个为 main 的日志格式
log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

access_log  logs/access.log  main;
```

上方的 `log_format` 后面类似 `$remote_addr` 是nginx的内置变量，取值如下

```
$remote_addr, $http_x_forwarded_for（反向） 记录客户端IP地址
$remote_user 记录客户端用户名称
$request 记录请求的URL和HTTP协议
$status 记录请求状态
$body_bytes_sent 发送给客户端的字节数，不包括响应头的大小； 该变量与Apache模块mod_log_config里
的“%B”参数兼容。
$bytes_sent 发送给客户端的总字节数。
$connection 连接的序列号。
$connection_requests 当前通过一个连接获得的请求数量。
$msec 日志写入时间。单位为秒，精度是毫秒。
$pipe 如果请求是通过HTTP流水线(pipelined)发送，pipe值为“p”，否则为“.”。
$http_referer 记录从哪个页面链接访问过来的
$http_user_agent 记录客户端浏览器相关信息
$request_length 请求的长度（包括请求行，请求头和请求正文）。
$request_time 请求处理时间，单位为秒，精度毫秒； 从读入客户端的第一个字节开始，直到把最后一个字符发送给客
户端后进行日志写入为止。
$time_iso8601 ISO8601标准格式下的本地时间。
$time_local 通用日志格式下的本地时间。
```

Nginx访问设置

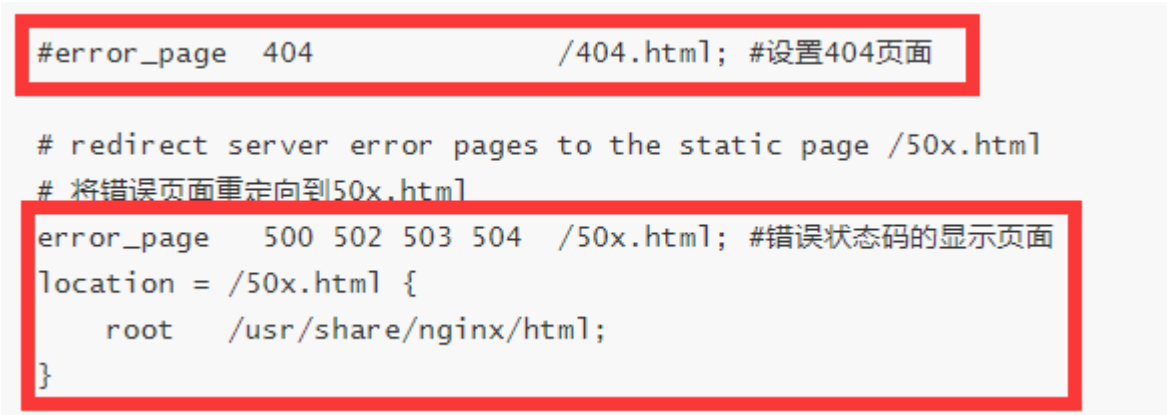
1、设置错误页面

打开nginx子默认配置文件 “default.conf”

```
vim /etc/nginx/conf.d/default.conf
```

error_page指令用于设置错误页面，如下图所示。500、502、503、504这些是常见的HTTP的错误代码，/50x.html用于当发生上述指定的任意一个错误的时候，使用网站根目录下的/50x.html进行处理。这里通过location = /50x.html进行匹配，最终用户看到的错误页面为 “/usr/share/nginx/html” 目录下的 50x.html页面。

设置格式：error_page ...



2、权限指令deny和allow

deny，设置禁止访问的IP

```
#禁止IP：192.168.6.101访问
location / {
    deny   192.168.6.101;
}

#禁止所有IP访问
location / {
    deny   all;
}
```

allow，设置允许访问的IP

```
#只允许IP：192.168.6.101访问
location / {
    allow  192.168.6.101;
}

#允许所有IP访问
location / {
    allow   all;
}
```

deny和allow的优先级

nginx的权限指令是从上往下执行的，在同一个块下deny和allow指令同时存在时，谁先匹配谁就会起作用，后面的权限指令就不会执行了。如下图，如果 “deny 111.111.111.111” 触发了，那它后面的两条都不会触发。

```
location / {
    deny 192.168.6.101;
    allow 192.168.6.220;
    deny 192.168.6.221;
}
```

其实他们的关系就像 “if...else if...else if”，谁先触发，谁起作用。

```
if (deny 192.168.6.101) {...}
else if (allow 192.168.6.220) {...}
else if (deny 192.168.6.221) {...}
```

Nginx虚拟主机配置

1、基于域名的虚拟主机配置

1、修改宿主机的hosts文件(系统盘/windows/system32/driver/etc/HOSTS)

linux： vim /etc/hosts

格式： ip地址 域名

eg: 192.168.3.172 www.gerry.com

2、在nginx.conf文件中配置server段

```
server {
    listen 80;
    server_name www.gerry.com; # 域名区分

    location / {
        root html/gerry;
        index index-1.html;
    }
}

server {
    listen 80;
    server_name www.mmren.com;  # 域名区分

    location / {
        root html/gerry;
        index index-2.html;
    }
}
```

2、基于端口号的虚拟主机配置

1、在nginx.conf文件中配置server段

```
server {
    listen 80; # 端口区分
    server_name www.gerry.com;

    location / {
        root html/gerry;
        index index.html;
    }
}

server {
    listen 8080; # 端口区分
    server_name www.gerry.com;

    location / {
        root html/gerry;
        index index.html;
    }
}
```

3、基于IP的虚拟主机配置

1、添加网卡的IP别名

```
ifconfig ens33:1 192.168.3.202 broadcast 192.168.3.255 netmask 255.255.255.0 up
route add -host 192.168.3.202 dev ens33:1
ifconfig ens33:2 192.168.3.203 broadcast 192.168.3.255 netmask 255.255.255.0 up
route add -host 192.168.3.203 dev ens33:2
```

从另外一台服务器Ping 192.168.3.202和192.168.3.203两个IP,如果能够Ping通，则证明配置无误。但是，通过ifconfig和route配置的IP别名在服 务器重启后会消失，不过可以将这两条ifconng和route命令添加到/etc/rc.local文件中，让系统开机时自动运行，以下是相关命令：**vi /etc/rc.local**
在文件末尾增加以下内容，然后保存即可

ifconfig ens33:1 192.168.3.202 broadcast 192.168.3.255 netmask 255.255.255.0 up
route add -host 192.168.3.202 dev ens33:1
ifconfig ens33:2 192.168.3.203 broadcast 192.168.3.255 netmask 255.255.255.0 up
route add -host 192.168.3.203 dev ens33:2

2、修改配置文件做如下的Server段配置

```
server {
    listen 80;
    server_name 192.168.3.202;

    location / {
        root html/host1;
        index index.html;
    }
}
```

```
server {
    listen 80;
    server_name 192.168.3.203;

    location / {
        root html/host2;
        index index.html;
    }
}
```

location匹配规则详解

语法规则：

```
# = 开头表示精准匹配
# ~ 大小写敏感# ~* 忽略大小写
# ^~ 只需匹配uri开头
# @ 定义一个命名的 location，在内部定向时使用，例如 error_page
location  [ = | ~ | ~* | ^~ ] /uri/ { ... }
location @name { ... }
```

1、 任意匹配

```
# 匹配所有请求，但是正则和最长字符串会优先匹配
location / {
    #规则
}
```

2、“=” 精准匹配

```
# 精确匹配 / ，可以匹配类似 `http://www.example.com/` 这种请求，'/'后不能带有任何内容
location = / {
   #规则
}
3.“~”大小写敏感
```

3、“~” 大小写敏感

```
location ~ /Test/ {
    #规则
}
#可以匹配     http://www.test.com/Test/
#不可以匹配     http://www.test.com/test/
```

4、 “~*” 大小写忽略

```
# ~* 会忽略uri部分的大小写
location ~* /Test/ {
    #规则
}
#可以匹配    http://www.test.com/test/
```

5、“^~” 只匹配uri开头

```
#以 /test/ 开头的请求，都可以匹配上
location ^~ /test/ {    #规则
}
#可以匹配    http://www.test.com/test/
```

6、“@” 命名匹配

```
#以 /img/ 开头的请求，如果链接的状态为 404。则会匹配到 @img_err 这条规则上。
location /img/ {
    error_page 404 @img_err;
}
location @img_err {
    # 规则
}
```

7、以内容结尾匹配

```
# 匹配所有以 gif,jpg或jpeg 结尾的请求
location ~* \.(gif|jpg|jpeg)$ {
  #规则
}
```

8、location匹配优先级

在配置中需要注意的一点就是location的匹配规则和优先级

- = 开头表示精确匹配
- ^~ 开头表示uri以某个常规字符串开头，不是正则匹配;
- ~ 开头表示区分大小写的正则匹配;
- ~* 开头表示不区分大小写的正则匹配;
- / 通用匹配, 如果没有其它匹配,任何请求都会匹配到;

9、location的匹配流程

A、判断是否精准匹配，如果匹配，直接返回结果并结束搜索匹配过程。
B、判断是否普通匹配，如果匹配，看是否包含^~前缀，包含则返回，否则记录匹配结果，(如果匹配到多个location时返回或记录最长匹配的那个)
C、判断是否正则匹配，按配置文件里的正则表达式的顺序，由上到下开始匹配，一旦匹配成功，直接返回结果，并结束搜索匹配过程。
D、如果正则匹配没有匹配到结果，则返回步骤B记录的匹配结果。

注意：

A、多个普通匹配的location时，和location的顺序无关，总是匹配所有的location，然后取匹配最长的location作为结果

B、多个正则匹配的location时，和顺序有关，从上到下依次匹配，一旦匹配成功便结束，直接返回结果。

常用配置指令alias、root

1、语法

- **root**

> 语法:root path
> 默认值:root html
> 配置段:http、server、location、if

- **alias**

> 语法:alias path
> 配置段:location

2、root&alias区别

2.1 root

> 如果一个请求的 URI 是/weblogs/test.html时,web 服务器将会返回服务器上的/data/weblogs/b.hundred.com/weblogs/test.html 的文件。

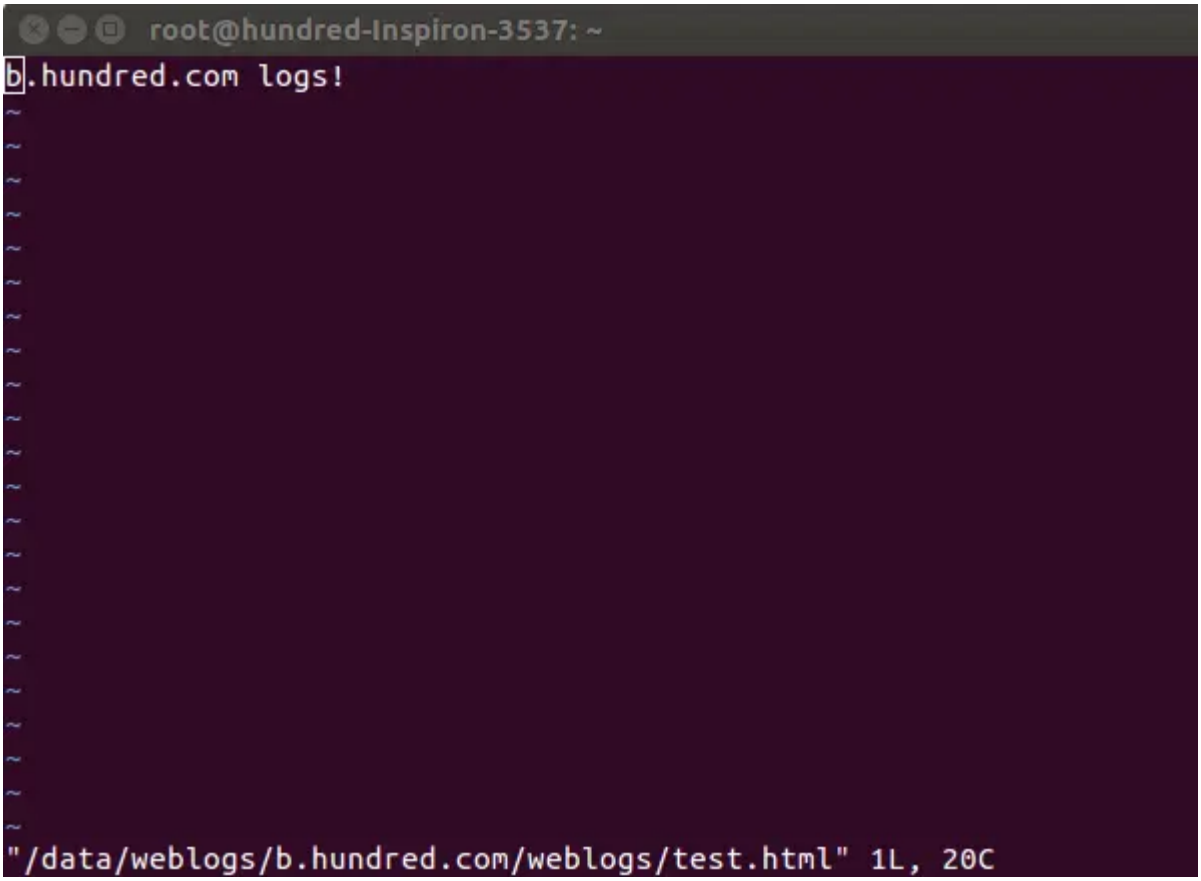
实例

```
server{
    listen 8081;
    server_name b.hundred.com;
    #root /data/site/b.hundred.com;
    access_log /data/logs/nginx/b.hundred.com-access.log main;
    location ~ ^/weblogs/ {
        root /data/weblogs/b.hundred.com;
        autoindex on;
        auth_basic  "Restricted";
        auth_basic_user_file passwd/weblogs;
    }
```

配置内容

```
root@hundred-Inspiron-3537:~# ls /data/weblogs/b.hundred.com/weblogs/
test.html
root@hundred-Inspiron-3537:~#
```

文件夹里的文件



test.html的内容

> 测试静态页面的路径是/data/weblogs/b.hundred.com/weblogs/test.html

浏览器请求

> - uri为/weblogs/test.html,请求后的页面是/data/weblogs/b.hundred.com/weblogs/mysite.html.
> **root 会根据完整的 URI 请求来映射,也就是/path/uri。**

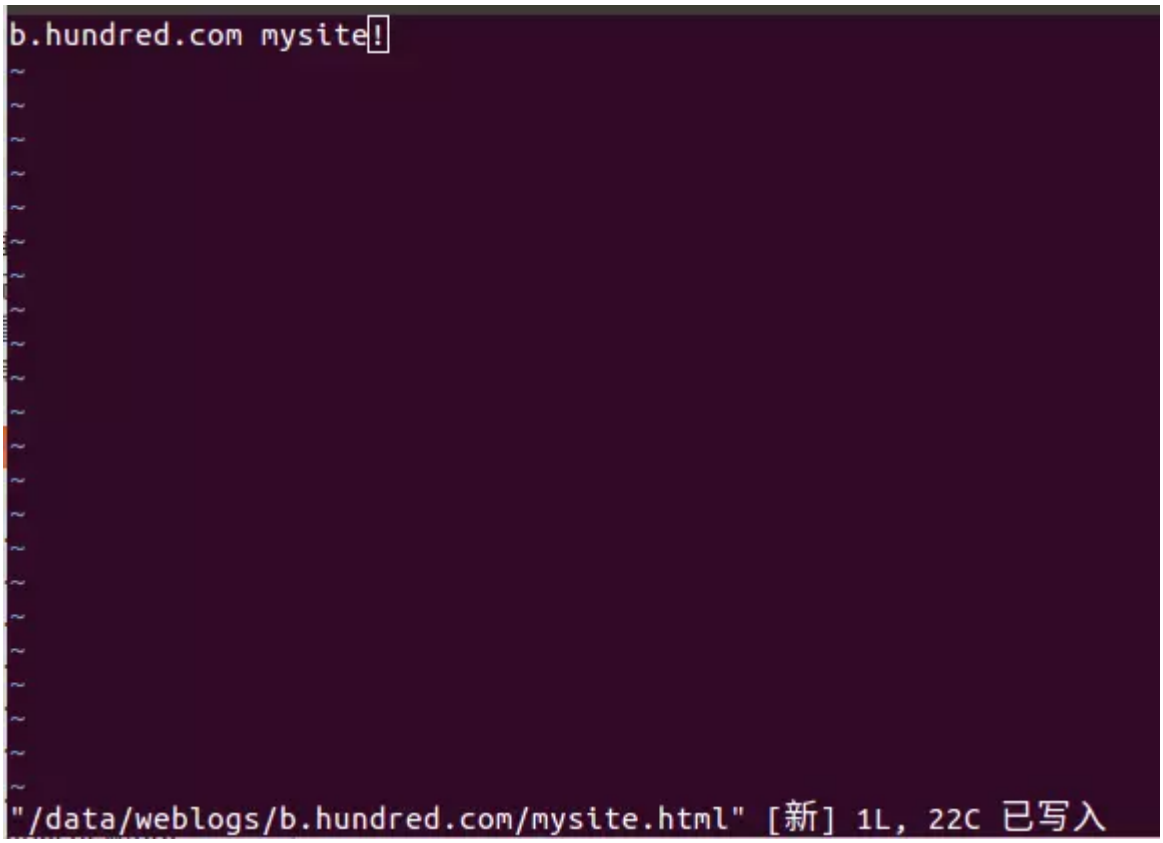
2.2 alias

> alias 会把 location 后面配置的路径丢弃掉,把当前匹配到的目录指向到指定的目录。

实例

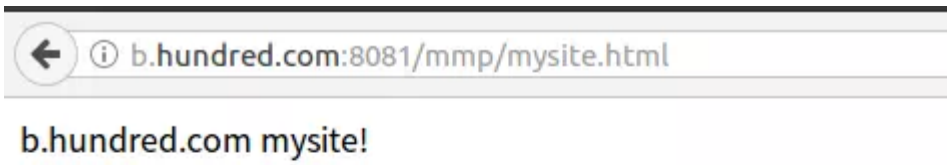
```
location ^~ /mmp/ {
    alias /data/weblogs/b.hundred.com;
    autoindex on;
    auth_basic  "Restricted";
    auth_basic_user_file passwd/weblogs;
}
```

配置内容



测试的静态文档

> 测试静态页面的路径是/data/weblogs/b.hundred.com/mysite.html



浏览器请求页面

> - uri为/mmp/mysite.html,请求后的页面是/data/weblogs/b.hundred.com/mysite.html.
> - /mmp/被替换为/data/weblogs/b.hundred.com/

注意

> - **使用 alias 时,目录名后面一定要加"/"。**
> - **alias 可以指定任何名称。**
> - **alias 在使用正则匹配时,必须捕捉要匹配的内容并在指定的内容处使用。**
> - **alias 只能位于 location 块中。**

2.3 实际操作时发生的问题

> 在使用alias配置时,使用了`alias /data/weblogs/b.hundred.com`,结果在浏览器发起请求时,找不到页面。原来是目录名后面没有加"/"。之后,改成了`alias /data/weblogs/b.hundred.com/`就成功了。

Nginx的Rewrite

1、rewrite的介绍

1、Rewrite通过ngx_http_rewrite_module模块支持url重写、支持if判断，但不支持else

2、rewrite功能就是，使用nginx提供的全局变量或自己设置的变量，结合正则表达式和标志位实现url重写以及重定向

3、rewrite只能放在server{},location{},if{}中，并且只能对域名后边的除去传递的参数外的字符串起作用

http://www.gerry.com/test/v1/10

http://www.gerry.com/test?version=1&id=10

2、常用指令

If 空格 (条件) {设定条件进行重写}

条件的语法：

1. “=” 来判断相等，用于字符比较
2. “~” 用正则来匹配（表示区分大小写），“~*” 不区分大小写
3. “-f -d -e” 来判断是否为文件、目录、是否存在

return 指令

语法：return code;

停止处理并返回指定状态码给客户端。

if ($request_uri ~ *.sh){

  return 403

}

set指令

set variable value;

定义一个变量并赋值，可以是文本、变量或者文本变量混合体

rewrite指令

语法：rewrite regex replacement [flag]{last / break/ redirect 返回临时301/ permanent 返回永久302}

- `last` : 相当于Apache的[L]标记，表示完成rewrite
- `break` : 停止执行当前虚拟主机的后续rewrite指令集
- `redirect` : 返回302临时重定向，地址栏会显示跳转后的地址
- `permanent` : 返回301永久重定向，地址栏会显示跳转后的地址

因为301和302不能简单的只返回状态码，还必须有重定向的URL，这就是return指令无法返回301,302的原因了。这里 last 和 break 区别有点难以理解：

1. last一般写在server和if中，而break一般使用在location中
2. last不终止*重写后*的url匹配，即新的url会再从server走一遍匹配流程，而break终止重写后的匹配
3. break和last都能阻止继续执行后面的rewrite指令

3、if指令与全局变量【实例】

if判断指令
语法为 `if(condition){...}`，对给定的条件condition进行判断。如果为真，大括号内的rewrite指令将被执行，if条件(conditon)可以是如下任何内容：

- 当表达式只是一个变量时，如果值为空或任何以0开头的字符串都会当做false
- 直接比较变量和内容时，使用 `=` 或 `!=`
- `~` 正则表达式匹配，`~*` 不区分大小写的匹配，`!~` 区分大小写的不匹配

`-f` 和 `!-f` 用来判断是否存在文件
`-d` 和 `!-d` 用来判断是否存在目录
`-e` 和 `!-e` 用来判断是否存在文件或目录
`-x` 和 `!-x` 用来判断文件是否可执行

例如：

```
if ($http_user_agent ~ MSIE) {
    rewrite ^(.*)$ /msie/$1 break;
} //如果UA包含"MSIE", rewrite请求到/msid/目录下

if ($http_cookie ~* "id=([^;]+)(?:;|$)") {
    set $id $1;
 } //如果cookie匹配正则，设置变量$id等于正则引用部分

if ($request_method = POST) {
    return 405;
} //如果提交方法为POST，则返回状态405（Method not allowed）。return不能返回301,302

if ($slow) {
    limit_rate 10k;
} //限速，$slow可以通过 set 指令设置

if (!-f $request_filename){
    break;
    proxy_pass  http://127.0.0.1;
} //如果请求的文件名不存在，则反向代理到localhost 。这里的break也是停止rewrite检查

if ($args ~ post=140){
    rewrite ^ http://example.com/ permanent;
} //如果query string中包含"post=140"，永久重定向到example.com

location ~* \.(gif|jpg|png|swf|flv)$ {
    valid_referers none blocked www.jefflei.com www.leizhenfang.com;
    if ($invalid_referer) {
        return 404;
    } //防盗链
}
```

全局变量

下面是可以用作if判断的全局变量

- `$args`： #这个变量等于请求行中的参数，同`$query_string`
- `$content_length`： 请求头中的Content-length字段。
- `$content_type`： 请求头中的Content-Type字段。
- `$document_root`： 当前请求在root指令中指定的值。
- `$host`： 请求主机头字段，否则为服务器名称。
- `$http_user_agent`： 客户端agent信息
- `$http_cookie`： 客户端cookie信息
- `$limit_rate`： 这个变量可以限制连接速率。
- `$request_method`： 客户端请求的动作，通常为GET或POST。
- `$remote_addr`： 客户端的IP地址。
- `$remote_port`： 客户端的端口。
- `$remote_user`： 已经经过Auth Basic Module验证的用户名。
- `$request_filename`： 当前请求的文件路径，由root或alias指令与URI请求生成。
- `$scheme`： HTTP方法（如http，https）。
- `$server_protocol`： 请求使用的协议，通常是HTTP/1.0或HTTP/1.1。
- `$server_addr`： 服务器地址，在完成一次系统调用后可以确定这个值。

- `$server_name`： 服务器名称。
- `$server_port`： 请求到达服务器的端口号。
- `$request_uri`： 包含请求参数的原始URI，不包含主机名，如：”/foo/bar.php?arg=baz”。
- `$uri`： 不带请求参数的当前URI，$uri不包含主机名，如”/foo/bar.html”。
- `$document_uri`： 与$uri相同。

例：`http://localhost:88/test1/test2/test.php`
：server_port：88
$request_uri：http://localhost:88/test1/test2/test.php
：document_root：/var/www/html
$request_filename：/var/www/html/test1/test2/test.php

4、常用正则

- `.`： 匹配除换行符以外的任意字符
- `?`： 重复0次或1次
- `+`： 重复1次或更多次
- `*`： 重复0次或更多次
- `\d`：匹配数字
- `^`： 匹配字符串的开始
- `$`： 匹配字符串的介绍
- `{n}`： 重复n次
- `{n,}`： 重复n次或更多次
- `[c]`： 匹配单个字符c
- `[a-z]`： 匹配a-z小写字母的任意一个
  http://www.rm.com/test/name/gerry
- http://www.rm.com/test/name/(.*) http://www.rm.com/test?name=$1
- http://www.rm.com/test?name=gerry

小括号`()`之间匹配的内容，可以在后面通过`$1`来引用，`$2`表示的是前面第二个`()`里的内容。正则里面容易让人困惑的是`\`转义特殊字符。

4.1、rewrite实例

例1：

```
http {
    # 定义image日志格式
    log_format imagelog '[$time_local] ' $image_file ' ' $image_type ' ' $body_bytes_sent '
' $status;
    # 开启重写日志
    rewrite_log on;

    server {
        root /home/www;

        location / {
                # 重写规则信息
                error_log logs/rewrite.log notice;
                # 注意这里要用‘’单引号引起来，避免{}
```

```
                http://www.rm.com/images/ab/9z9z2/bm.jpg
                http://www.rm.com/data?file=bm.jpg
                rewrite '^/images/([a-z]{2})/([a-zO-9]{5})/(.*)\.(png|jpg|gif)$' /data?
file=$3.$4;
                # 注意不能在上面这条规则后面加上“last”参数，否则下面的set指令不会执行
                set $image_file $3;
                set $image_type $4;
        }

        location /data {
                # 指定针对图片的日志格式，来分析图片类型和大小
                access_log logs/images.log mian;
                root /data/images;
                # 应用前面定义的变量。判断首先文件在不在，不在再判断目录在不在，如果还不在就跳转到最后一个
url里
                try_files /$arg_file /image404.html;
        }
        location = /image404.html {
                # 图片不存在返回特定的信息
                return 404 "image not found\n";
        }
}
```

对形如 `/images/ef/uh7b3/test.png` 的请求，重写到 `/data?file=test.png`，于是匹配到 `location /data`，先看 `/data/images/test.png` 文件存不存在，如果存在则正常响应，如果不存在则重写tryfiles到新的image404 location，直接返回404状态码。

例2:

```
rewrite ^/images/(.*)_(\d+)x(\d+)\.(png|jpg|gif)$ /resizer/$1.$4?width=$2&height=$3? last;
```

对形如 `/images/bla_500x400.jpg` 的文件请求，重写到 `/resizer/bla.jpg?width=500&height=400` 地址，并会继续尝试匹配location。

```
server {
  listen 8888;
  server_name www.rm.com;
  default_type text/html;

  location = /ok {
    return 200 "ok";
  }

  location = /redirect {
    return 302 http://www.baidu.com;
  }

  location / {
    rewrite /test1/(.*) http://www.$1.com;
    return 200 "ok";
  }
```

```
  #location / {
  #  rewrite /test1/(.*) www.$1.com;
  #  return 200 "ok";
  #}

  location / {
    rewrite ^/test1 /test2;
    rewrite ^/test2 /more/index.html break;
    rewrite /more/index.html /test4;
    proxy_pass http://www.baidu.com;
  }

  location = /test2 {
    return 200 "test2";
  }

  location = /test3 {
    return 200 "test3";
  }

  location = /test4 {
    return 200 "test4";
  }
}
```

5、项目实例讲解

执门搜索

这个规则的目的是为了执行搜索，搜索URL中包含的关键字。

```
加了问号和不加问号区别?
URI: www.rm.com:8888/test/gerry?name=zhangsan

/search.do?name=gerry&name=zhangsan            不加问号。

/search.do?name=gerry                           加问号。
```

热门搜索

```
请求的URL         //hqidi.com/search/gerry?name=zhangsan
重写后URL         //hqidi.com/search.do?name=gerry
重写规则          rewrite ^/search/(.*)$ /search.do?name=$1?;
```

用户个人资料页面

大多数运行访问者注册的动态网站都提供一个可以查看个人资料的页面，这个页面的URL包含用户的UID和用户名

```
请求的URL         //hqidi.com/user/47/dige
重写后URL         //hqidi.com/user.do?id=47&name=dige
重写规则          rewrite ^/user/([0-9]+)/(.+)$ /user.do?id=$1&name=$2?;
```

多个参数

有些网站对字符串参数使用不同的语法，例如 通过斜线"/"来分隔非命名参数

```
请求的URL  //hqidi.com/index.php/param1/param2/param3
重写后URL  //hqidi.com/index.php?p1=param1&p2=param2&p3=param3
重写规则            rewrite ^/index.do/(.*)/(.*)/(.*)$ /index.do?p1=$1&p2=$2&p3=$3?;
```

类似百科的格式

这种格式特点，一个前缀目录，后跟文章名称

```
请求的URL  //hqidi.com/wiki/some-keywords
重写后URL  //hqidi.com/wiki/index.do?title=some-keywords
重写规则            rewrite ^/wiki/(.*)$ /wiki/index.do?title=$1?;
```

论坛

论坛一般用到两个参数，一个话题标识(topic)一个出发点(starting post)

```
请求的URL  //hqidi.com/topic-1234-50-some-keywords.html
重写后URL  //hqidi.com/viewtopic.do?topic=1234&start=50
重写规则            rewrite ^/topic-([0-9]+)-([0-9]+)-(.*)\.html$ viewtopic.do?
topic=$1&start=$2?;
```

新网站的文章

这种URL结构的特点，由一个文章标识符，后跟一个斜线，和一个关键字列表组成。

```
请求的URL  //hqidi.com/88/future
重写后URL  //hqidi.com/atricle.php?id=88
重写规则            rewrite ^/([0-9]+)/.*$ /aticle.do?id=$1?;
```

最后一个问号

若被替换的URI中含有参数(类似/app/test.php?id=5之类的URI)，默认情况下参数会被自动附加到替换串上，可以通过在替换串的末尾加上?标记来解决这一问题。

```
rewrite ^/users/(.*)$ /show?user=$1? last;
```

比较一个加上？标记和不加？标记的URL跳转区别：

```
rewrite ^/test(.*)$ //hqidi.com/home premanent;
```

访问//hqidi.com/test?id=5 经过301跳转后的URL地址为 //hqidi.com/home?id=5

```
rewrite ^/test(.*)$ //hqidi.com/home? premanent;
```

访问//hqidi.com/test?id=5 经过301跳转后的URL地址为 //hqidi.com/home

Nginx的代理实战

1、Nginx的正向代理

建议拿两台服务器去测试，效果比较直观，现在我给服务器A设置正向代理，服务器B设置拦截，假设服务器A的IP地址是 192.168.6.121，服务器B的IP地址是192.168.6.220；

先来配置服务器A

```
server{
    resolver 8.8.8.8; # dns解析 8.8.8.8/114.114.114.114/62.128.128.68
    listen 80;
    server_name www.gerry.com 192.168.6.220;
    location / {
        proxy_pass http://$http_host$request_uri;
    }
}
```

再去配置服务器B

```
server{
    server_name www.rm.com;
    listen 80;

    location / {
        #注意这里if和括号中间有一个空 不然保存会报错
        if ( $remote_addr !~* "192\.168\.6\.121" ){
            return 403;
        }

        root html;
        index index.html;
    }
}
```

配置完成服务器A和B的nginx后，重启nginx服务 `/usr/local/nginx/conf/nginx.conf -s reload`

这里保存服务器B的时候可能会报错，修改保存的指令为 `/usr/local/nginx/sbin/nginx -c /usr/local/nginx/conf/nginx.conf`

然后在本地去尝试访问服务器B的80端口，会返回一个403的错误页面，说明拦截已经成功了，再来验证正向代理成功了没，在geogle浏览器里下载一个扩展插件 Proxy SwitchySharp，把网页的代理手动设置为服务器A的IP和80端口，设置成功以后切换到服务器A的代理，再去访问服务器B的80端口发现访问成功了，说明正向代理已经成功！

2、Nginx的反向代理

在浏览器输入 *www.abc.com* , 从 nginx 服务器跳转到 linux 系统 tomcat 主页面。

```
server {
        listen       80;
        server_name  www.gerry.com;   #监听地址

        location  / {
           root html;  #/html目录
           proxy_pass http://127.0.0.1:8080;  #请求转向
           index  index.html index.htm;      #设置默认页
        }
    }
```

根据在浏览器输入的路径不同，跳转到不同端口的服务中。

```
server {
        listen       80;
        server_name  www.gerry.com;   #监听地址

        location  ~ /example1/ {
           proxy_pass http://192.168.6.220:8088;
        }

        location  ~ /example2/ {
           proxy_pass http://192.168.6.220:8089;
        }
    }
```

proxy_pass 既可以是ip地址，也可以是域名，同时还可以指定端口

3.1 反向代理常用指令

补充：

```
#设置代理地址
proxy_pass https://www.baidu.com/

#设置请求头-并将头信息传递到服务器端，不属于请求头的参数中也需要传递时，重定义下即可
proxy_set_header Host $host:$server_port; #服务器名称和端口一起通过代理服务器传递

#配置nginx与代理服务器尝试建立连接的超时时间，单位秒
proxy_connect_timeout 300

#设置与代理服务器的读超时时间，单位秒
proxy_read_timeout 300

#配置服务器数据回传时间，单位秒
proxy_send_timeout 300

#指定修改被代理服务器返回的响应头中的location头域跟refresh头域数值
proxy_redirect off
```

负载均衡

*upstream*是Nginx的HTTP Upstream模块，这个模块通过一个简单的调度算法来实现客户端IP到后端服务器的负载均衡

1、Upstream常用参数介绍

语法：server address [parameters]

其中关键字server必选。
address也必选，可以是主机名、域名、ip或unix socket，也可以指定端口号。
parameters是可选参数，可以是如下参数：

down：表示当前server已停用

backup：表示当前server是备用服务器，只有其它非backup后端服务器都挂掉了或者很忙才会分配到请求

weight：表示当前server负载权重，权重越大被请求几率越大。默认是1

max_fails和**fail_timeout**一般会关联使用，如果某台server在fail_timeout时间内出现了max_fails次连接失败，那么Nginx会认为其已经挂掉了，从而在fail_timeout时间内不再去请求它，fail_timeout默认是10s，max_fails默认是1，即默认情况是只要发生错误就认为服务器挂掉了，如果将max_fails设置为0，则表示取消这项检查。

```
upstream tomcatserver {
    server 192.168.3.220:8080;
    server 192.168.3.222:8080 weigth=4;
}
```

在做负载均衡前，我们首先需要定义一个 Server 组用来表示所有存在的后台服务：

```
 http {
     upstream backend {
         server backend1.example.com weight=5;
         server backend2.example.com;
         server 192.0.0.1 backup;
     }
 }
```

然后，我们需要把流量重定向到上一步定义的 backend 上去, 我们可以通过指定 proxy_pass 的值来完成这一操作：

```
     upstream backend {
         server backend1.example.com;
         server backend2.example.com;
         server 192.0.0.1 backup;
     }

     server {
         location / {
             proxy_pass http://backend;
         }
     }
 }
```

这里我们将所有的流量重定向到 http://backend , 这将这个 NGINX 实例上的所有流量重定向到之前定义的 backend 上去。

负载均衡算法

当没有指定任何信息时， NGINX 默认使用了 Round Robin(轮询)算法来重定向流量。其实 NGINX 提供了多种算法来做负载均衡，下面我们来介绍一下：

1、Round Robin (轮询)

在没有指定 weight(权重) 的情况下，Round Robin 会将所有请求均匀地分发给所有后台服务实例：

```
upstream backend {
    server backend1.example.com;
    server backend2.example.com;
}
```

这里我们没有指定权重，所以两个后台服务会收到等量的请求。但是，当指定了权重之后，NGINX 就会将权重考虑在内：

```
upstream backend {
    server backend1.example.com weight=5;
    server backend2.example.com;
}
```

在 NGINX 中，weight 默认被设置为 1。这里我们用一开始的配置举例， backend1.example.com 的权重被设置为 5，另一个的权重没设置，所以是默认值 1。我们也没有设置轮询算法，所以这时候 NGINX 会以 5：1 的比例转发请求，即 6 个请求中， 5 个被放到了 backend1.example.com 上， 有一个被发到了 backend2.example.com 上。

2、Least Connections（最少连接算法）

在这个模式下，一个请求会被 NGINX 转发到当前活跃请求数量最少的服务实例上：

```
upstream backend {
    least_conn;
    server backend1.example.com;
    server backend2.example.com;
}
```

我们用 least_conn 来指定最少连接优先算法, NGINX 会优先转发请求到现有连接数少的那一个服务实例上。

3、IP Hash (IP 哈希)

在 IP Hash 模式下，NGINX 会根据发送请求的 IP 地址的 hash 值来决定将这个请求转发给哪个后端服务实例。被 hash 的 IP 地址要么是 IPv4 地址的前三个 16 进制数或者是整个 IPv6 地址。使用这个模式的负载均衡模式可以保证来自同一个 IP 的请求被转发到同一个服务实例上。当然，这种方法在某一个后端实例发生故障时候会导致一些节点的访问出现问题。

```
 upstream backend {
     ip_hash;
     server backend1.example.com;
     server backend2.example.com;
 }
```

如果某一台服务器出现故障或者无法进行服务，我们可以给它标记上 down，这样之前被转发到这台服务器上的请求就会重新进行 hash 计算并转发到新的服务实例上:

```
upstream backend {
    server backend1.example.com;
    server backend2.example.com;
    server backend3.example.com down;
}
```

NGINX 提供了多种负载均衡模式，在实际使用中，需要根据实际业务需求去做尝试，分析日志来找到最适合当前场景的复杂均衡模式。

Nginx 缓存

1、 案例实战

实现效果:
　　在3天内，通过浏览器地址栏访问 *http://192.168.6.220/images/a.jpg* ，不会从服务器抓取资源，3天后（过期）则从服务器重新下载。
具体配置:

```
# http 区域下添加缓存区配置
proxy_cache_path /tmp/nginx_proxy_cache levels=1 keys_zone=cache_one:512m inactive=60s
max_size=1000m;

# server 区域下添加缓存配置
location ~ \.(gif|jpg|png|htm|html|css|js)(.*) {
     proxy_pass http://192.168.6.220:8088; #如果没有缓存则转向请求
     proxy_redirect off;
     proxy_cache cache_one;
     proxy_cache_valid 200 1h;            #对不同的 HTTP 状态码设置不同的缓存时间
     proxy_cache_valid 500 1d;
     proxy_cache_valid any 1m;
     expires 3d;
}
```

expires 是给一个资源设定一个过期时间，通过 expires 参数设置，可以使浏览器缓存过期时间之前的内容，减少与服务器之间的请求和流量。也就是说无需去服务端验证，直接通过浏览器自身确认是否过期即可，所以不会产生额外的流量。此种方法非常适合不经常变动的资源。

动静分离

1、案例实战

实现效果：

通过浏览器访问http://www.abc.com/index.jsp

如果请求路径为.jsp结尾将会访问动态资源服务器dynamic

如果请求路径为.css/.js/.html结尾的资源，首先判断你指定的静态资源目录下面是否存在请求的资源，如果存在直接访问响应，如果不存在就到静态资源服务器或者资源

具体配置：

```
upstream static {
    server www.abc.com:8089;
}

upstream dynamic {
    server www.abc.com:8088;
}

server {
    listen       80;
    server_name  www.rm.com;

    # 拦截动态资源
    location ~ .*\.(php|jsp)$ {
       proxy_pass http://dynamic;
    }

    # 拦截静态资源
    location ~ .*\.(jpg|png|htm|html|css|js)$ {
       root /data/;
       ## 判断请求的文件是否存在，不存在就反向代理到静态服务器读取数据
       try_files $uri @static_proxy;
    }

    location @static_proxy {
        proxy_pass http://static;
    }
}
```

nginx跨域

跨域的解决办法就是在`header`里面加上允许跨域的源等信息

```
server {
           listen 80;
           server_name www.gerry.com;

           location /pc {
               proxy_pass http://localhost:8081/;

                proxy_set_header Host $host;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
```

```
                proxy_set_header X-Forwarded-Proto $scheme;
                proxy_set_header X-Forwarded-Port $server_port;
            }


            location / {
                if ($request_method = 'OPTIONS') {
                    add_header Access-Control-Allow-Origin $http_origin always;
                    add_header Access-Control-Allow-Credentials true always;
                    add_header Access-Control-Allow-Methods 'GET,POST,PUT,DELETE,OPTIONS'
always;
                    add_header Access-Control-Allow-Headers 'Authorization,X-Requested-
With,Content-Type,Origin,Accept' always;
                    add_header Access-Control-Max-Age 3600;
                    add_header Content-Length 0;
                    return 200;
                }

                add_header Access-Control-Allow-Origin $http_origin always;
                add_header Access-Control-Allow-Credentials true always;
                add_header Access-Control-Allow-Methods 'GET,POST,PUT,DELETE,OPTIONS'
always;
                add_header Access-Control-Allow-Headers 'Authorization,X-Requested-
With,Content-Type,Origin,Accept' always;

                proxy_pass http://localhost:8081/;
            }
        }
```

但是在实际项目里。`origin`还是不要设置为`*`比较好，因为前端使用axios的话在获取session这一块会出现问题。

参数说明：

1、Access-Control-Allow-Origin，这里使用变量 $http_origin取得当前来源域，大家说用“*”代表允许所有，我实际使用并不成功，原因未知；

2、Access-Control-Allow-Credentials，为 true 的时候指请求时可带上Cookie，自己按情况配置吧；

3、Access-Control-Allow-Methods，OPTIONS一定要有的，另外一般也就GET和POST，如果你有其它的也可加进去；

4、Access-Control-Allow-Headers，这个要注意，里面一定要包含自定义的http头字段（就是说前端请求接口时，如果在http头里加了自定义的字段，这里配置一定要写上相应的字段），从上面可看到我写的比较长，我在网上搜索一些常用的写进去了，里面有“web-token”和“app-token”，这个是我项目里前端请求时设置的，所以我在这里要写上；

5、Access-Control-Expose-Headers，可不设置，看网上大致意思是默认只能获返回头的6个基本字段，要获取其它额外的，先在这设置才能获取它；

6、语句“ if ($request_method = 'OPTIONS') { ”，因为浏览器判断是否允许跨域时会先往后端发一个 options 请求，然后根据返回的结果判断是否允许跨域请求，所以这里单独判断这个请求，然后直接返回；

Gzip压缩

gzip是网页的一种压缩技术，经过gzip压缩后，页面大小可以变为原来的30%甚至更小。gzip网页压缩的实现需要浏览器和服务器的支持，实际上就是服务器端压缩，传到浏览器后浏览器解压并解析。

1、gzip的配置说明

```
#是否开启gzip模块，on表示开启，off表示关闭
gzip on;

#设置允许压缩的页面最小字节(从header头的Content-Length中获取)
gzip_min_length 1k;

#设置gzip申请内存的大小，其作用是按块大小的倍数申请内存空间，param2:int(k) 后面单位是k。
#这里设置以16k为单位，按照原始数据大小以16k为单位的4倍申请内存
#如果没有设置，默认值是申请跟原始数据相同大小的内存空间去存储gzip压缩结果
gzip_buffers 4 16k;

#识别http协议的版本
gzip_http_version 1.1;

#设置gzip压缩等级，等级越底压缩速度越快文件压缩比越小，反之速度越慢文件压缩比越大
#等级1-9，9 压缩比最大但处理最慢（传输快但比较消耗cpu）
gzip_comp_level 2;

#设置需要压缩的MIME类型,非设置值不进行压缩，即匹配压缩类型
gzip_types text/plain application/x-javascript text/css application/xml;

#用于在响应消息头中添加Vary：Accept-Encoding
#使代理服务器根据请求头中的Accept-Encoding识别是否启用gzip压缩
gzip_vary on;
```

打开nginx全局配置文件

```
vim /etc/nginx/nginx.conf复制代码
```

2、设置gzip压缩

```
http {
    ...
    gzip  on;
    gzip_types text/plain text/html  application/javascript text/css;
    ...
}
```

为了方便看出设置gzip压缩后的效果，百度搜索 “网页Gzip压缩检测”，分别记录压缩前和压缩后的检测结果。。

Nginx的HTTPS配置

1、HTTPS简介

> 1.https简介

> HTTPS其实是有两部分组成：HTTP + SSL / TLS，也就是在HTTP上又加了一层处理加密信息的模块。服务端和客户端的信息传输都会通过TLS进行加密，所以传输的数据都是加密后的数据
>
> 2.https协议原理
>
> 首先，客户端与服务器建立连接，各自生成私钥和公钥，是不同的。服务器返给客户端一个公钥，然后客户端拿着这个公钥把要搜索的东西加密，称之为密文，并连并自己的公钥一起返回给服务器，服务器拿着自己的私钥解密密文，然后把响应到的数据用客户端的公钥加密，返回给客户端，客户端拿着自己的私钥解密密文，把数据呈现出来

2、开启nginx的ssl模块

```
1.the "ssl" parameter requires ngx_http_ssl_module  in /usr/local/nginx/conf/nginx.conf:37
原因是nginx缺少http_ssl_module模块，编译安装时带上--with-http_ssl_module配置就可以了
2.如果已经安装过nginx,想要添加模块看下面
1)切换到nginx源码包
cd /usr/local/src/nginx-1.11.3
2)查看ngixn原有的模块
/usr/local/nginx/sbin/nginx -V
3)重新配置
./configure --prefix=/usr/local/nginx --with-http_stub_status_module --with-http_ssl_module
4)重新编译，不需要make  install安装。否则会覆盖
make
5)备份原有已经安装好的nginx
cp /usr/local/nginx/sbin/nginx /usr/local/nginx/sbin/nginx.bak
6)将刚刚编译好的nginx覆盖掉原来的nginx(ngixn必须停止)
cp ./objs/nginx /usr/local/nginx/sbin/
这时，会提示是否覆盖，请输入yes，直接回车默认不覆盖
7)启动nginx，查看nginx模块，发现已经添加
/usr/local/nginx/sbin/nginx -V
```

3、证书和私钥的生成

```
注意：一般生成的目录,应该放在nginx/conf/ssl目录

1.创建服务器证书密钥文件 server.key:
openssl genrsa -des3 -out server.key 1024

输入密码，确认密码，自己随便定义，但是要记住，后面会用到。

2.创建服务器证书的申请文件 server.csr

openssl req -new -key server.key -out server.csr

输出内容为:
Enter pass phrase for root.key: ← 输入前面创建的密码
Country Name (2 letter code) [AU]:CN ← 国家代号，中国输入CN
State or Province Name (full name) [Some-State]:BeiJing ← 省的全名，拼音
Locality Name (eg, city) []:BeiJing ← 市的全名，拼音
Organization Name (eg, company) [Internet Widgits Pty Ltd]:MyCompany Corp. ← 公司英文名
```

```
Organizational Unit Name (eg, section) []: ← 可以不输入
Common Name (eg, YOUR name) []: ← 此时不输入
Email Address []:admin@mycompany.com ← 电子邮箱，可随意填
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []: ← 可以不输入
An optional company name []: ← 可以不输入

4.备份一份服务器密钥文件

cp server.key server.key.org

5.去除文件口令
openssl rsa -in server.key.org -out server.key

6.生成证书文件server.crt
openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

4、配置文件

proxyPort=443

redirectPort=443

```
server{
       #比起默认的80 使用了443 默认 是ssl方式  多出default之后的ssl
       listen 443 default ssl;
       #default 可省略
       #开启  如果把ssl on；这行去掉，ssl写在443端口后面。这样http和https的链接都可以用
       ssl on;
       #证书(公钥.发送到客户端的)
       ssl_certificate ssl/server.crt;
       #私钥,
       ssl_certificate_key ssl/server.key;
       #下面是绑定域名
        server_name www.daj.com;
        location / {
            #禁止跳转
            proxy_redirect off;
            #代理淘宝
            proxy_pass https://www.gerry.com;
        }
}
```

重启Nginx

Nginx配置udp/tcp代理

1、安装模块

```
./configure --prefix=/usr/local/nginx  --with-stream  --with-http_stub_status_module
```

2、配置文件

```
#nginx.conf部分配置
# upd/tcp
stream {
    upstream backend {
        server 192.168.3.173:3306;
    }
    server {
        listen 8686;
        proxy_connect_timeout 8s;
        proxy_timeout 24h;   #代理超时
        proxy_pass backend;
    }
}

http {

}
```

Nginx双机主备架构搭建实战

1、安装keepalived

Keepalived安装

> 利用keepalived做主备，避免单点问题，实现高可用
> 在 `192.168.1.219` 和 `192.168.1.220` 做主备，前者主，后者备

- 安装Keepalived

```
[root@rabbit1 tmp] yum -y install keepalived
```

- 配置Keepalived生成VIP

```
[root@rabbit1 tmp] vim /etc/keepalived/keepalived.conf
```

部分配置信息（只显示使用到的）：

```
global_defs {
   # 路由id,主备节点不能相同
   router_id node1

}

vrrp_script chk_haproxy {
    script "/etc/keepalived/nginx_check.sh"
    interval 5
    weight 10
}
```

```
 vrrp_instance VI_1 {
     # BACKUP 表示备份节点
     state BACKUP
     interface enp0s8
     virtual_router_id 1
     # 优先级，备份节点要比主节点低
     priority 50
     advert_int 1
     authentication {
         auth_type PASS
         auth_pass 123456
     }

     track_script {
         chk_nginx
     }

     virtual_ipaddress {
         192.168.0.200
     }
 }
```

nginx检查脚本

```
#!/bin/bash
# 判断haproxy是否已经启动
if [ `ps -C nginx --no-header |wc -l` -eq 0 ] ; then
    #如果没有启动，则启动
    /usr/local/nginx/sbin/nginx
fi

#睡眠3秒以便haproxy完全启动
sleep 3

#如果haproxy还是没有启动，此时需要将本机的keepalived服务停掉，以便让VIP自动漂移到另外一台haproxy
if [ `ps -C nginx --no-header |wc -l` -eq 0 ] ; then
    systemctl stop keepalived
fi
```

创建后为其赋予执行权限：

```
chmod +x /etc/keepalived/nginx_check.sh
```

> 发现脚本执行一直出现127问题解决：
>
> 临时关闭：
>
> [root@localhost ~]# getenforce
> Enforcing

> [root@localhost ~]# setenforce 0
> [root@localhost ~]# getenforce
> Permissive
>
> 永久关闭：
>
> [root@localhost ~]# vim /etc/sysconfig/selinux
>
> SELINUX=enforcing 改为 SELINUX=disabled
>
> 重启服务reboot

2、安装nginx